International Journal of Medical Informatics (2004) **73**, 75—89

International Journal of Medical Informatics

www.intl.elsevierhealth.com/journals/ijmi

Towards a continuous evolution and adaptation of information systems in healthcare

R. Lenz*, K.A. Kuhn

Institute of Medical Informatics, Philipps-University Marburg, Bunsenstrasse 3, D-35037 Marburg, Germany

Received 4 April 2003; received in revised form 26 July 2003; accepted 29 September 2003

KEYWORDS
Information systems;
Information management;
Software;
Quality assurance;
Health care

Summary *Objectives:* To address the problem of alignment of health information systems to healthcare processes, which is a major challenge in healthcare organizations; to present a layered approach for system evolution and adaptation based on an application framework and rapid application development; to accomplish a demand-driven system evolution by embedding the software engineering process in business process optimization projects and by closely involving end users to improve their own work practices. *Methods:* We have used a holistic health information system as a core application framework. System functionality is incrementally improved using an integrated ''generator tool'' for rapid application development. We have developed an iterative and participatory software engineering process, adapted to the conditions of the generator tool. The documentation techniques provided by the Unified Modeling Language (UML) were modified to achieve a straight forward documentation covering the whole development cycle from the business process model to generator-based computer applications. *Results and conclusion:* The layered approach for system evolution did provide an environment in which a flexible and participatory software development process could be established. Today, generator-based applications are used in all clinical departments of our 1200-bed University Hospital. We expect that tools for rapid application development will be further improved and will play an increasingly important role to establish responsive IT-infrastructures where the application developer can concentrate on business process alignment instead of coding and debugging.

1. Introduction

Information Technology is typically seen as a key enabler to improving healthcare processes due to its potential of providing rapid and comprehensive access to information at the point of care. Healthcare processes, however, are subject to change. Both internal causes (e.g. new diagnostic or therapeutic procedures, change of departmental structure) as well as external causes (e.g. introduction of DRGs, economic pressure towards hospital mergers, and healthcare integrated networks) enforce process changes and create a need to rapidly adapt health information systems to the new conditions. Moreover, Berg and Toussaint have pointed out that the organizational processes emerging during information system deployment are highly unpredictable, and that information systems 'requirements' will *necessarily* evolve in this process [1]. To some degree, our own project experiences have confirmed

*Corresponding author. Tel.: +49-6421-286-66298; fax: +49-6421-286-63599.
E-mail address: lenzr@mailer.uni-marburg.de (R. Lenz).

this [2]. Consequently, a responsive IT infrastructure is required to be able to rapidly and adequately react to newly arising needs [3,4]. This includes the capability of an IT infrastructure to add new functionality with minimal risk—but this is clearly a non-trivial task. IT projects are highly complex and still show an unacceptable high tendency to result in various facets of failure [5]. Sauer has identified not less than 50 factors for project failures which can be categorized into different factor classes, comprising user involvement, mutual understanding of project participants, benefit/value of outcome, management commitment, technical design quality and system complexity, project management, system and project performance, resource adequacy, situational stability, and implementation process issues including change management and requirements management [5]. These factors are highly interdependent and sometimes conflicting: Elimination of single, known failure factors is too simplistic and has not been successful. Identifying these factors, however, did deliver valuable hints to improve Software Engineering (SE) methodologies [6—8]. Considering how the scope of IT projects has changed over the years might help to understand why failure rates did not decrease, despite newly arising powerful technologies, such as client server, object orientation, and component technologies, which have fueled expectations to open up new application areas, and, with each stage, the core of what has been considered robust has been extended [5]. The dynamics of capability and opportunity have led to technology-driven innovation instead of business-demand-driven innovation [5]. In the 1980s and 1990s business process reengineering (BPR) has been seen as the magic method to achieve dramatic improvements by radically throwing away the old and starting over anew using the capabilities of the new technology [9,10]. BPR projects, however, have continued to fail at high rates from 50% to 80% [3,11,12]. Smith and Fingar pointed out that because it lacks any empirical foundation, ''reengineering'' lost credibility in the world of business, and they recommended to shift towards a new rule for business process management, where process owners design and deploy their own processes [3]. This is consistent with observations made in the healthcare domain: Berg and Toussaint have warned to rashly replace medical work practices which have evolved over a long period of time, and which are already adapted to the working needs of healthcare professionals [1]. Atkinson and Peel have recommended to transform a hospital through growing, not building, an electronic patient record system [13]. The central question is *how* to incrementally evolve and adapt an IT system to changing process requirements without introducing new problems on the technical level. How can we achieve an alignment of business process improvement and IT system evolution? How does a ''responsive IT infrastructure'' look like?

In this article we present a layered approach for system evolution based on an application framework and an integrated CASE tool (Computer-Aided Software Engineering). The rationale behind this approach is to support rapid application development and adaptation to the evolving needs for process improvement within a demand-driven software development process with very short iteration cycles.

In the background section, we motivate the potential of application frameworks and tool-based software development, and we summarize major principles of modern software engineering processes, such as iterative programming and user participation. To adequately consider these principles, it is necessary to adapt the software engineering process to the conditions of a concrete development tool and the underlying design primitives. In the subsequent sections we explain this adaptation, utilizing a holistic health information system and an integrated CASE tool to establish a layered approach for system evolution.

2. Background

Domain specific application frameworks are a promising approach to foster software reuse and system evolution. They are used as a basis for system development with the intention to save development costs by providing reusable skeletons on which new applications can be built [14]. The object-oriented framework approach [14,15] promises to dramatically reduce the cost of building new (object oriented) applications. Building such a framework that can serve as the underlying skeleton for a group of applications is a demanding task. Moreover, once a framework has been built, generating a new application by specializing it appropriately also requires considerable expertise. A high level rapid application development (RAD) toolkit [16] can be used to ease the usage of an application framework. The intention of using such a RAD tool is to shorten development cycles and to bring the application developer closer to the end user. The application developer should be able to concentrate more on requirements engineering and adaptation to the end users needs instead of coding and debugging.

To really make effective use of a RAD tool and improve software quality, an adapted software engineering process is required as a guideline for tool-based application development. Software engineering is aimed at the application of a systematic, disciplined, and quantifiable approach that covers the whole lifecycle of software. The traditional waterfall model is inadequate for RAD, as it does not consider that requirements change during the software development process [17]. Traditional SE methods also tend to long development cycles with late feedback by users, resulting in the risk of failure and of raising costs. Particularly, in the healthcare domain, many authors recommend a highly participatory software development process to achieve well adapted and accepted applications that fit to the healthcare professionals' work practice and into their ''business processes'' [18,19].

Iterative SE models like the Rational Unified Process (RUP®) [6,7] and Extreme Programming (XP) [8] try to prevent the ''late design breakage'' by continuously considering end user feedback. The RUP is based on software-based practices which have successfully contributed to the production of high quality software in industry: to develop software iteratively, to manage requirements, to use component-based architectures, to visually model software, to continuously verify software quality, and to control changes to software [7]. By using the Unified Modeling Language (UML) [20], the state of the art methodology for documenting object-oriented software development, a common understanding of all tasks, responsibilities, and artifacts is facilitated. UML diagrams and RUP recommendations form a comprehensive catalog, which needs to be adapted to the specific requirements and circumstances of particular software development projects [7]. Iteration, without doubt, is a must. A central question, however, is when to build a prototype and when to build the operative system? The use of prototyping is critical, as there is a fine line between gaining knowledge and losing resources in terms of time and money ([15], p. 626). Prototyping is less costly in smaller projects, which make shorter iteration cycles, closer user participation, and early feedback much easier: this results in better adaptation to real world requirements, better user acceptance, and reduction of risk. XP [8] is a software development approach which radically focuses on incremental improvement by very short iteration cycles (minutes and hours instead of weeks and months), continuous testing and user feedback, simple design, and code review (e.g. by programming in pairs). Agile practices, such as XP, are increasingly transforming methods and methodologies for developing information systems. Yet, it is still unclear how to benefit from agile practices in the development of large complex information systems. What kind of documentation is suitable for XP programming, and what kind of tool support is helpful? XP is particularly designed for building comparatively small systems in an environment of changing requirements. Healthcare institutions do represent an environment of changing requirements, but health information systems (HIS) are typically huge and complex systems. How can we handle this complexity and still achieve flexible adaptation and continuous system evolution?

To answer this question it is important to recognize that it is simply not necessary to comprehensively map the information and knowledge management process in healthcare to an IT system — physicians, nurses, and patients also accumulate knowledge and communicate with each other, while IT systems only complement and support these processes. Berg and Toussaint have described the nature of medical work practice as inherently sketchy and ad hoc, and the models underlying patient records as highly partial and eclectic [1]. Thus, IT systems' evolution should be prepared to continuously react on these partial and even transient models, and new (partial) models should be added as it appears appropriate to improve clinical work practice. Agile software development techniques perfectly fit with the idea of incrementally improving an information system by adding or modifying partial views. Consequently, we need to establish an infrastructure which effectively enables such incremental evolution of the overall system by creation of small subsystems embedded in an overall system, which can be developed independently in small and manageable projects.

3. A layered approach for system evolution

A reduction of complexity is of paramount importance to improve development and evolution of information systems. Layered approaches have been used successfully to reduce complexity and achieve manageability in different traditional areas of computer science such as operating systems [21], database systems [22], and communication systems [23]. Based on our project experiences with a health information system [2], we suggest to transfer this principle to IS development and evolution to achieve both flexibility and robustness. The idea is to utilize a core application framework with relatively stable and robust functionality, which is supportive to higher layers providing tools for quick and adaptive application development. The resulting layered approach for information systems evolution with different roles and system

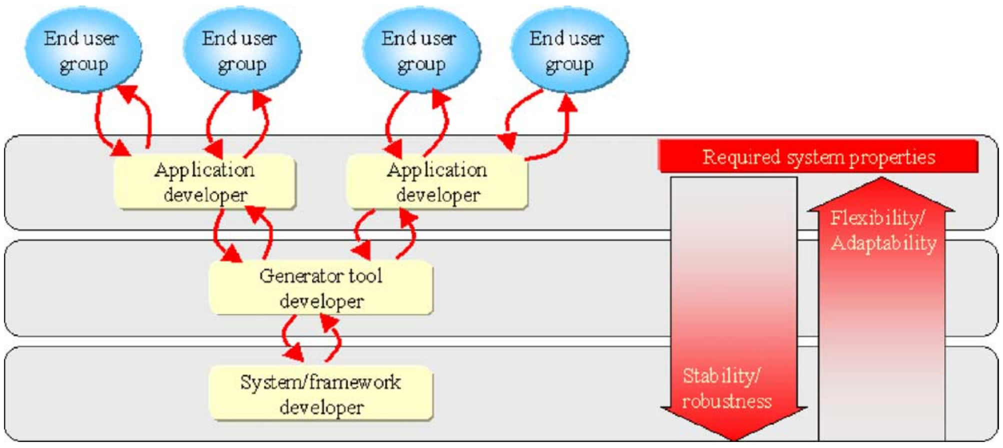

Fig. 1 Different roles in a layered IT system evolution approach: the application developer uses the generator tool to rapidly implement applications that are adapted to the end users' needs. He receives his feedback primarily from the end users. Software engineering on this level must cope with changing requirements and business demands. Thus, flexibility and adaptability are core requirements for the applications to be generated. The generator tool developer receives his feedback from the application developer, who reports on desideratum to improve the usability and effectivity of the tool. The system developer is responsible for the underlying application framework, which provides the skeleton of objects and functions that are to be reused in new applications. The desired system properties at this layer differ greatly from those at the top layer: As software bugs in the lower system levels tend to propagate to multiple applications on the top level, robustness and stability should have the highest priority of system properties at this level. Nevertheless, performance is also a major issue which is of equal importance.

properties is outlined in Fig. 1. In this three-layer model, we distinguish different roles accordingly:

1. The *framework developer* is responsible for the overall system architecture and the core system functionality which provides the common basis for all applications.
2. The *generator tool developer* is responsible for providing a programming environment (generator tool) that eases the reuse of the underlying (domain specific) infrastructure provided by the application framework.
3. The *application developer* utilizes the generator tool to create new applications and adapt existing applications to specific requirements.

The application programmers, at the top layer, are developing integrated applications with very short iteration cycles without the need to involve the lower level roles in each project. Flexibility and adaptability have to be the most important system properties of the top level. At the architectural basis, the lower levels are required to provide stability and robustness in the first place. Modifications and improvements will also become necessary at the lower levels, but to ensure stability, iteration cycles should be significantly longer. In addition, modifications at the lower levels have to preserve the validity of applications on the upper levels. In this article, we concentrate on the software engineering process at the top level of this hierarchy, which is located at the intersection of systems evolution and business process management.

The rationale of the approach is that application developers do not necessarily need low level encoding skills. Instead, they need domain knowledge as well as social and communicative skills.

4. Methods and tools

The SE process described in this article has been adapted on the basis of a health information system which is in widespread use in Germany, Austria, and Switzerland: the Orbis®/OpenMed-system. The installation in Marburg represents one of the most advanced installations of this system, and significant impulses concerning the development, improvement, and use of an integrated RAD tool (generator tool) have resulted from the Marburg project, which has been described in [2,24].

The generator tool is part of a vendor-specific application framework which allows to incrementally add new application modules or adapt generic applications to specific requirements (Fig. 2). Workflow is supported on the basis of a typical document-oriented workflow paradigm, e.g. see [25]. The tool is used by developers of the vendor company, while a nearly identical version of the tool is being used by IT personnel of several larger healthcare institutions. Subsequently we will simply refer to the users of the generator tool as ''application developers''.

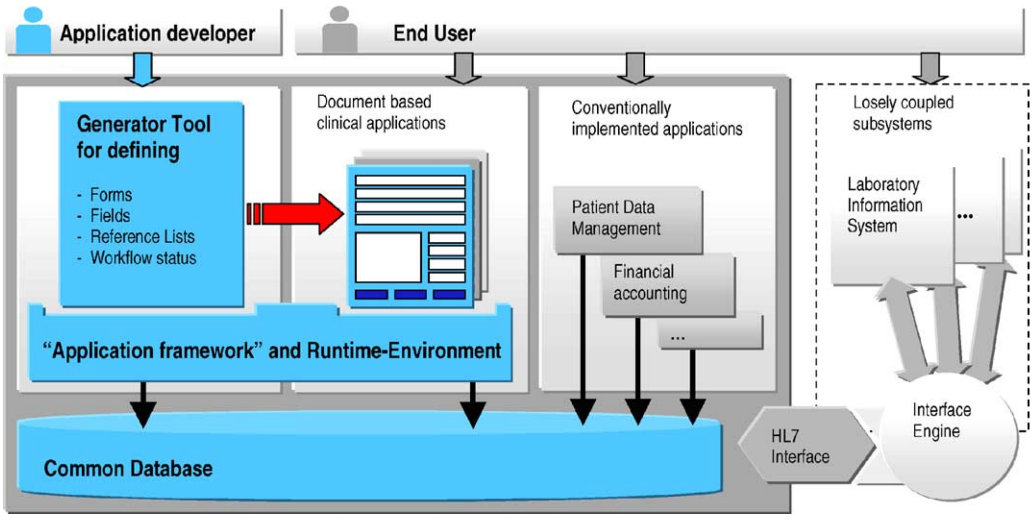
Application developer
End User
Generator Tool for defining
- Forms
- Fields
- Reference Lists
- Workflow status
Document based clinical applications
Conventionally implemented applications
Losely coupled subsystems
Patient Data Management
Financial accounting
...
Laboratory Information System
...
"Application framework" and Runtime-Environment
Common Database
HL7 Interface
Interface Engine

Fig. 2 System architecture—overview [24]. Conventional administrative applications as well as document-based clinical applications directly operate on a common central database. Document-based applications are embedded into an application framework which provides access to commonly used objects. A generator tool is used to incrementally add new document-based applications to the system. The application developer who uses the generator tool creates a new application by defining forms with dedicated fields, which are mapped to commonly used objects. In addition, a document flow is established by defining reference lists and selectively assigning forms to these lists according to their current workflow status.

The interaction paradigm of the generator tool is motivated by the traditional paper-based clinical work practice. The application programmer uses the tool to specify paper-like electronic documentation forms. Just like a paper form, an instance of an electronic form (electronic document) is filled in incrementally and sent to a particular destination to either trigger a procedure or to document results. Intra- and inter-departmental document flow is supported by selectively displaying an electronic document in different ''reference lists'' or ''work lists''(e.g. list of reports generated, list of reports validated, etc.), according to its current workflow status. The underlying application framework facilitates data integration via a common database, and the generator tool provides means to specify which data are to be automatically uploaded into a specific form (for further details see [24]). The tool can be used to parameterize pre-existing form templates, e.g. a standard discharge letter can be adapted to the needs of a specific department. However, the tool can also be used to develop completely new applications, which are fully integrated with the rest of the system from the very beginning.

Programming with a generator tool like this is different from traditional programming (e.g. object-oriented programming), as application developers do not directly edit the code of an application. Instead they primarily deal with higher level design primitives like forms, fields, lists of electronic documents (''work lists''), and so on. The SE process, in particular the documentation of tool-generated applications, has to be adapted to this kind of programming.

5. An adapted software engineering process

A major goal of adapting SE methodologies to the conditions of a generator tool is to align business process analysis with the specification of generator-based IT applications. Accordingly, UML documentation techniques were adapted to fulfill two requirements:

- Some kind of process documentation (process model) is required to support interdisciplinary communication and foster a common understanding of relevant issues. In particular, the process model should describe the business context of the IT application.
- The process model should be suitable for directly deriving the functional specification of a generator-based application.

Use cases have proven to be a good technique for capturing the relevant process-related information within structured interviews. The result is a description of business processes in form *of business use cases* and a *business use case dia-*

gram. Business use cases, however, only present a functional view on business processes but are not suitable for modeling the dynamic aspects of processes. UML activity diagrams can be used to model the business context in which an IT application is to be used [20]. They show the flow of control and objects (e.g. documents) between different activities. However, activity diagrams are somewhat limited in expressing typical aspects of clinical processes such as different roles involved in *one* activity, resources to be used by an activity, or location where an activity takes place. Activity diagrams also tend to lose clearness if one activity uses more than three or four objects. To avoid these disadvantages we developed a tool for clinical process documentation, which is mainly based on UML activity diagrams but contains additional aspects of clinical processes. The tool MapDoc (*Ma*rburg *p*rocess *doc*umentation tool) and its underlying methodology have been used within several projects for process analysis and optimization in different departments of the Marburg university hospital [2]. A MapDoc business process model can be seen as an activity diagram where each activity represents a particular business use case. Thus, the process model can also be interpreted as the ''top level use case''. The main constituents of a MapDoc Activity are shown in Fig. 3.

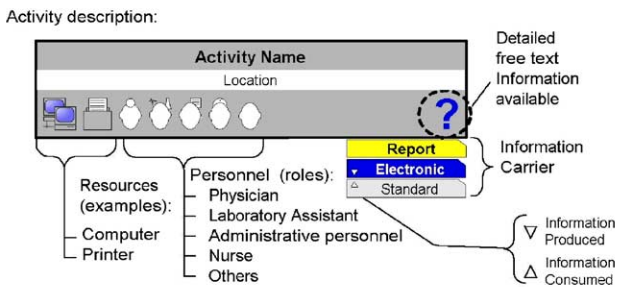

Fig. 3 A MapDoc Activity. The constituents of a MapDoc Activity are: the name of the activity, the location where it takes place, the resources needed (e.g. a computer), the responsible personnel, and information carriers. Predefined roles for personnel are physician, nurse, laboratory assistant, administrative personnel, and others. For each of these roles additional distinctions can be defined (e.g. medical director or assistant medical director as additional differentiation of physician). Information carriers can be both computer applications or conventional paper forms. The visual representation of an activity is complemented by a detailed free text description of the purpose of the activity and by additional information about the contents of the information carriers. The contents of information carriers are described by ''information blocks'' (e.g. patient demographic data). Information flow is documented by defining which information blocks are produced and which are consumed by an activity.

A MapDoc process model is built within several steps:

1. *Roles*. All involved user groups are categorized into different roles. The tool provides predefined symbols for physicians, nurses, laboratory personnel, administrative personnel and others. Arbitrary roles can be defined and visually represented by these symbols.
2. *Locations*. Where do activities take place? All relevant locations are named in order to be consistently referenced by different activities.
3. *Resources*. The tool provides a catalog of symbols for typical resources (e.g. computer, printer, different medical devices, etc.) needed in clinical processes. The catalog can be extended by simply adding new named symbols.
4. *Information carriers and objects*. Information carriers are needed as the containers for information flow. The contents of an information carrier (objects) can be defined in arbitrary depth (object hierarchy); content definition is not mandatory, though. Typically, an information carrier contains different ''information blocks'' (e.g. ''patient demographic data'', ''diagnosis'', ''ADT data'', etc.). The distinction of relatively high level information blocks is usually sufficient to specify information flow. A detailed description of the contents of these information blocks is done in a class diagram which is used to describe the underlying ontology.
5. *Activities, flow of control, and flow of information*. A process model is an extended activity diagram, where each activity is described by a name, the location where it takes place, the roles of the responsible personnel involved, the resources needed within the activity, and the information carriers needed for the activity. The flow of control is specified as in a UML activity diagram. To specify information flow, it is necessary to define which information blocks in which information carrier are used as input and which are used as output for a particular activity.
6. *Events*. Events are used to describe synchronization points needed for a complete description of business processes (e.g. ''patient arrives'') (cf. Fig. 4).
7. *Additional symbols and comments*. Some additional symbols can be used to express frequently occurring phenomena. For example, a clock symbol is used to express that an activity is not directly triggered by the completion of its predecessor (cf. Fig. 4). In addition, arbitrary comments can be added to a process model, if there are important process aspects that are not covered by the available modeling primitives.

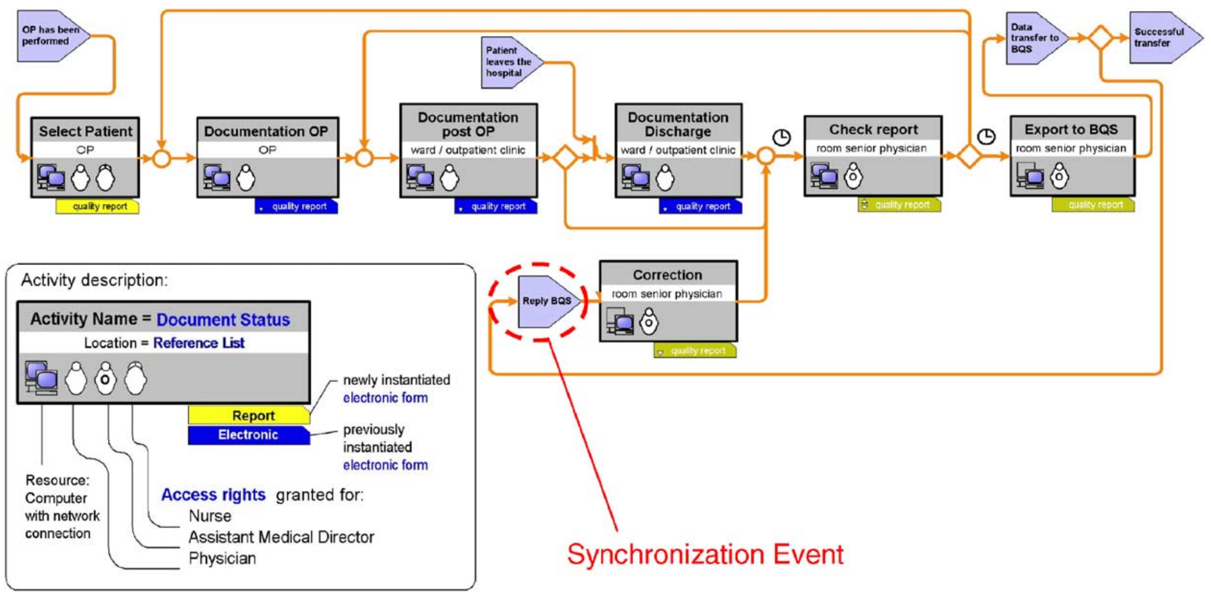

Fig. 4 MapDoc target process model for an electronic form to support quality assurance in gynecology. Each activity represents a different status in the workflow of the electronic document to be designed.

A target process model describes how a document-based application is embedded into the healthcare process (cf. Fig. 4). A conceptual specification of a tool-based application comprising access rights, workflow states, and required work lists can be directly derived from this model in a straight forward way.

6. A UML description of the adapted SE process

MapDoc is a specialized process documentation tool which is focused on describing *clinical* processes and preparing the implementation of computer applications that support these processes. MapDoc diagrams are part of the documentation which is produced during the SE process. The *SE process itself* is described by means of UML activity diagrams as proposed in [26]. The main process (Fig. 5) shows how software development is interwoven with clinical process analysis and improvement. The embedded process for implementing work packages is shown in Fig. 6. The main phases of the software engineering process are described subsequently.

6.1. Domain exploration

Each project starts with an exploration of the application domain, where the preconditions for project success are verified or established. These include management support and involvement of key users [27]. The key user concept has proven to be a practicable way to achieve close end user involvement without slowing down project progress too much. Yet, all user groups are to be involved into the project to some extent in order to achieve ownership and avoid resistance [27].

6.2. Requirements analysis

Once a project has been established, structured interviews are used to capture business use cases and describe the status quo within an initial process model. As the whole task of process improvement is demand-driven, process modeling should also be demand-driven, which means the initial process model should represent a quick reference for the relevant aspects of the projects' scope. A glossary is defined and continuously extended to provide reference definitions for relevant terms (see e.g. [7], p. 67). Existing documentation is collected, analyzed, and used as a basis for a class diagram which describes the information objects that are relevant in the applications' scope. The process model and class diagrams are checked in cooperation with end users, and suggestions for process improvements are collected. In this stage, detection and analysis of existing organizational problems are important, as an insufficient organization will impose a risk of failure, even if the project is only intended to adapt an application which has already been successfully introduced in other departments of the same institution. So, organizational changes may be required prior to an IT intervention. Typical indicators for organizational deficiencies, such

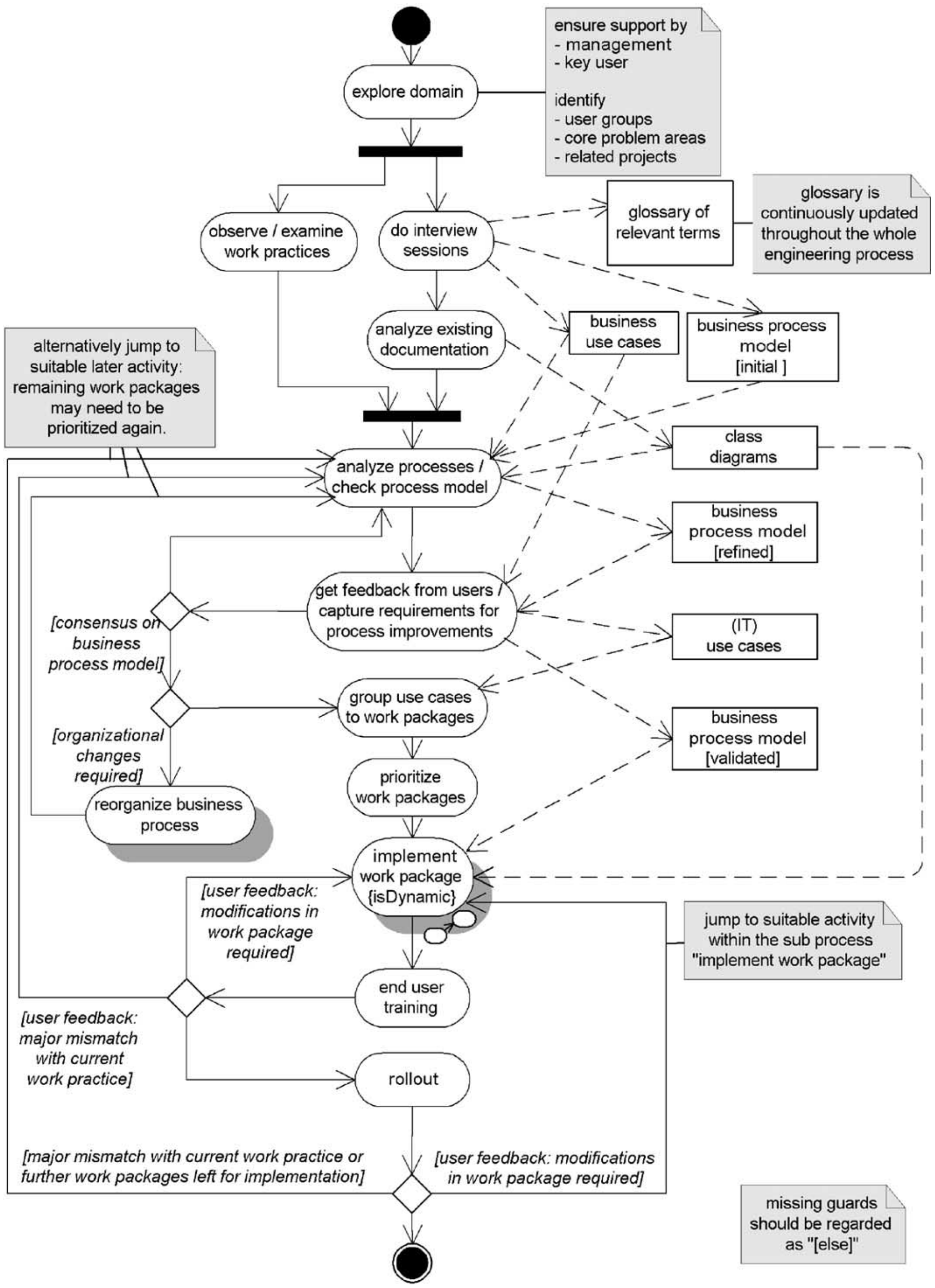
explore domain
ensure support by
- management
- key user
identify
- user groups
- core problem areas
- related projects
observe / examine work practices
do interview sessions
glossary of relevant terms
glossary is continuously updated throughout the whole engineering process
analyze existing documentation
business use cases
business process model [initial]
alternatively jump to suitable later activity: remaining work packages may need to be prioritized again.
class diagrams
analyze processes / check process model
business process model [refined]
get feedback from users / capture requirements for process improvements
[consensus on business process model]
(IT) use cases
group use cases to work packages
[organizational changes required]
business process model [validated]
reorganize business process
prioritize work packages
implement work package {isDynamic}
[user feedback: modifications in work package required]
jump to suitable activity within the sub process "implement work package"
end user training
[user feedback: major mismatch with current work practice]
rollout
[major mismatch with current work practice or further work packages left for implementation]
[user feedback: modifications in work package required]
missing guards should be regarded as "[else]"

Fig. 5 Main process: Software engineering is embedded in business process improvement process.

as waiting times, missing information, unused resources, etc. are checked. In our experience, the process model has been helpful in this stage to achieve a common understanding of the process. Organizational deficiencies became apparent by discussing the process with different participants. A process model is not a panacea for finding organizational deficiencies, but a thorough understanding of the different process steps is a precondition for an effective process optimization. If it turns out that the reasons for organizational deficiencies are more complex or that the effects of potential measurements are not clear, additional techniques and tools (e.g. process simulation [28,29] or FMEA failure mode and effects analysis [30,31]) should be used in addition to static process diagrams. This

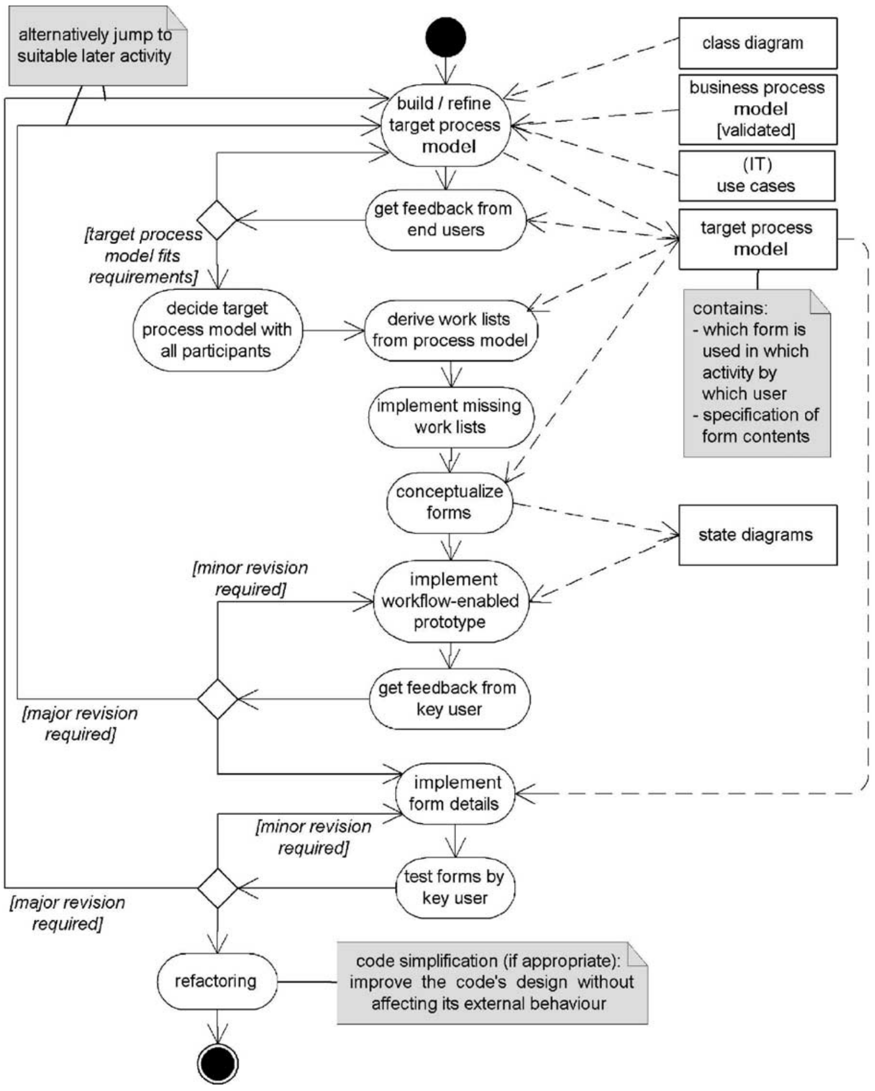
alternatively jump to suitable later activity
build / refine target process model
class diagram
business process model [validated]
(IT) use cases
target process model
get feedback from end users
[target process model fits requirements]
decide target process model with all participants
derive work lists from process model
contains:
- which form is used in which activity by which user
- specification of form contents
implement missing work lists
conceptualize forms
state diagrams
[minor revision required]
implement workflow-enabled prototype
get feedback from key user
[major revision required]
implement form details
[minor revision required]
test forms by key user
[major revision required]
refactoring
code simplification (if appropriate): improve the code's design without affecting its external behaviour

Fig. 6 Sub process: ''Implement work package''—participatory development of form-based applications.

will slow down project progress, but in some cases it can be helpful to assess different alternative approaches for process quality improvement prior to implementation.

6.3. Building work packages and prioritizing IT use cases

If the organization is ready for an IT intervention, the IT use cases can be directly derived from the process model. For each activity to be supported by the new IT application, one IT use case is defined. The end users decide which use cases and functionality are important and which features are less important. The application developer estimates the effort needed to implement each use case. IT use cases are then grouped to work packages which are prioritized in cooperation with end users according to their importance and feasibility. Each work package is shaped to be independently applicable, so that the overall system can be improved step by step by implementing different work packages. The highest priority work package is selected for implementation.

6.4. Conceptual design

A target process model is derived by iteratively refining the initial business process model together with end users until consensus is reached. Fig. 4 shows an example of such a (conceptual) target process model. This model describes how an electronic form is embedded into the healthcare processes, which information blocks are produced and consumed during this process, and which users (roles) are accessing the form at which stage of the process. This information is sufficient to directly derive the workflow specification of an electronic form. Each activity in the process model that uses an electronic document stands for a different document state. Thus, a state diagram can be derived specifying the possible state changes of an electronic document, and work list(s) can be assigned to this diagram, specifying where the document appears according to its current state. Access rights are derived from the roles assigned to activities in the target process model.

6.5. Implementation of work packages

Electronic forms are implemented by iteratively improving prototypes. A two-step approach has proven to be feasible: In the first step workflow is implemented according to the corresponding state diagram (a comprehensive example is given in [2]). In this stage form contents are indicated by placing dummy fields on the form, which are not yet connected to the central database. Provisionary forms without real data are sufficient to help users decide whether the flow model fits their work practice. Form contents are implemented in the second step if the workflow has proven to be usable. To achieve integration with the overall HIS it is important to reuse existing information which is already stored in the central database (e.g. patient information and available reports can be automatically uploaded into a discharge letter). The generator tool supports this by providing an object hierarchy as a reference for connecting fields to the central database. In addition to this, sub-forms can be reused that directly operate on the central database. This means that the application programmer usually refers to an additional level of abstraction instead of directly accessing the database. This has at least two major advantages: first, the programmer is not confronted with the whole complexity of the database schema, and second, database schema modifications that might come with release changes do not directly affect generator-based applications.

For data to be entered into electronic forms, it is important to make data entry compatible with departmental or hospital-wide coding conventions. End users can decide whether they want terminological control for a specific field or not. In case terminological control is desired, the application programmer can provide this by attaching an appropriate catalog (or a coding application) to this field. The contents of this catalog are either filled by some standard (e.g. ICD, ICPM, etc.) or by the users themselves (departmental or hospital specific catalogs).

6.6. Testing and iterative refinement

Functionality and usability of readily implemented forms are tested by the key user prior to end user training and rollout. The application developer thereby observes the key user as he or she performs tasks and logs any problems. The relatively short iteration cycles and the separate verification of workflow allow to use incremental prototypes instead of throw-away prototypes at low risk. Iterations are triggered by end user feedback and are possible at any stage of the process. After some iterations it might be recommendable to simplify the code to keep it readable and maintainable (''refactoring'') [8].

7. Generator-based software development in practice

While the general SE process presented is essentially not dependent on a specific tool, it has been elaborated and refined in our specific HIS environment [2]. Since 1999, generator-based applications have been used in all clinical departments of our 1200-bed University Hospital. An integrated clinical workstation based on the generator technology is in use at all wards throughout the hospital and in the outpatient clinics. ICD and ICPM codes are captured in a structured way via generator-based applications throughout the hospital. Generator-based discharge letters are in use at 17 out of 26 clinical departments, both in inpatient and outpatient settings. A generator-based scheduling application is run by seven outpatient clinics. Report generation and documentation in 10 ancillary departments (including radiology) is supported by generator-based applications. By the integrative approach, reports from these ancillary departments can be directly used in discharge letters which are written at wards or in clinics. Currently, the central database contains about 95 Mio single document entries and about 1,600,000 documents (for details of the database schema see [24]). Large catalogues of diagnoses and procedures are managed by the various applications. The system is being continuously improved by incrementally adding new generator-based applications to the running system and adapting generic applications to specific departmental needs.

Two examples out of numerous applications developed by our hospital IT staff are the implementation of a pathology documentation system [2] and of an endoscopy documentation system. Both applications support inter- and intra-departmental workflow (result reporting), while other systems include order entry functionality, too. The pathology system includes three documentation forms for diagnoses, result reporting, and billing and work lists (e.g. different work lists for macroscopical reporting, microscopical reporting, and final signature etc.). The endoscopy system covers one documentation form and three work lists. In both systems, reports are written in free text, and additional domain-specific vocabularies of low complexity are used, e.g. for indications, diagnoses, interventions, devices, locations, etc. ICD 10 coding is used in both applications. For both systems, about 10 person-days of IT personnel were needed for requirement analyses, implementation, and training of users. During the first 2—3 weeks of routine use, about 2—3 person-days (about 2/3 IT personnel, 1/3 physician time) per week were needed for trouble-shooting, user support, specification, and implementation of modifications.

Today, several applications developed at our hospital are being used by the vendor company as the basis for generic modules that are parameterized and adapted for use in other hospitals. Some further examples for IT projects that arose from demand-driven incremental improvement of business processes are:

- The optimization of pre-operative autologous blood donation (PABD). PABD is the collection of a patient's own blood prior to elective surgery to be used for his or her own transfusion needs. In the Marburg University Medical Center, PABD is an interdisciplinary process between the clinic for orthopaedic surgery and the institute for transfusion medicine. As the process coordination was found to be inefficient, a process analysis was initiated. A process model was developed as the basis for interdisciplinary communication and process optimization. In addition to the static process model an FMEA analysis was performed to determine potential risks of reorganizing the process. By eliminating redundant activities and reorganizing the workflow the number of activities could be reduced from 15 to 7. After the successful process redesign, the optimized workflow could be supported electronically by two simple electronic forms: one for acknowledgement of a performed PABD and the other that served as an indication checklist for orthopedists.
- The adaptation of discharge letters is one example out of numerous projects, where an existing application has been substantially improved by adding small but valuable features which helped to improve practitioners' work practice. In this case an existing workflow-enabled discharge letter has been improved by providing diagnoses and findings from previous visits for selection and adoption into the new discharge letter.

The vendor company uses the generator tool for developing generic clinical modules for deployment in different healthcare facilities. These modules are tested and cooperatively improved in our hospital to enhance adaptation to clinical work practice. Checklists for testing have been developed and are continuously extended as new bugs become apparent.

8. Summary of experiences

The layered approach for system evolution has been used in various demand-driven projects at the Marburg University Medical Center. The project scope has ranged from simply parameterizing existing applications to implementing complete departmental systems. The UML modeling techniques were modified to achieve a straight forward documentation covering the whole development cycle from the business process model to the system documentation. Use cases, extended activity diagrams (MapDoc diagrams), class diagrams, and state diagrams were useful as a basis for interdisciplinary communication. Process modeling was found to be particularly helpful in projects where an organizational redesign was within the project scope. In small projects, where the generator tool was only used for parameterization of existing applications, process modeling was often perceived as a waste of time. However, as a target process model is not only a means to achieve a common understanding but also provides a documentation of the application, it is still desirable to keep the model up to date to keep the system maintainable.

The layered approach for system evolution did provide an environment in which a flexible iterative and participatory software development process could be established. To our experiences, the generator and the underlying application framework used in Marburg can and will be improved in several ways:

- In its current version, the generator tool used in Marburg still has a major drawback that hinders its seamless integration into a semi-formal software engineering process: It does not support documentation. The MapDoc tool for documenting business processes has been designed to reflect the artifacts needed for generator-based applications, but it is not integrated with the generator tool. In practice this has led to diverging process models and implementation.

- In a layered approach, the lower system levels hide the complexity of generic functionality to the higher levels. Unfortunately, system bugs in the lower system layers often become apparent at the application layer where debugging is difficult, as the application programmer only sees higher level design primitives which hide the underlying code. Such bugs tend to magnify their effect, as generic services are typically reused in multiple tool-generated applications. Thus, there is a high risk if framework stability and robustness is insufficient.

  Problems concerning robustness of the core application framework did also slow down project progress in early stages of the Marburg project, which can be mostly attributed to the fact that Marburg was the first large scale implementation of the generator approach. In the meantime robustness has been continuously improved and more emphasis is put on this aspect.
- Another typical effect of using a RAD tool for system development is its reduced expressiveness: application developers have been limited by the predefined design primitives of the generator tool. If there are requirements that do not fit into this schema, implementation becomes more cumbersome and not as straightforward as described in our SE process. The experiences within the Marburg project have led to various improvements of the generator tool's expressiveness. Currently a major new release is being implemented, supporting graphical specification of workflow.

Despite these deficiencies, the advantages of database integration and the potential to easily support cross-departmental workflow did pay off. Information has become easily available within the holistic system: Hospital-wide, result reports from ancillary departments, discharge letters written under the generator environment, and ICD codes have become available without the need for any interface mapping [2,32], and order entry has been introduced.

9. Discussion

The layered approach for system evolution as presented in this article is basically a holistic approach where each software layer reuses common functionality of the underlying layers. Loosely coupled component-based systems offer an alternative approach for system evolution by interfacing ''best-of-breed'' components from different vendors (e.g. see [33—35]). To achieve data consistency within such a composed system it is important to carefully plan the overall system architecture, and to interface subsystems to central components like a master patient index. Connectivity on the technical level has been substantially improved by component technologies (e.g. see [36,37]) which provide an *infrastructure* for transparent access to distributed and technically heterogeneous components. Enterprise Application Integration (EAI) tools—like integration engines—typically provide mediator services that ease the management of heterogeneous interfaces. The really hard integration problem, however, is to bridge *semantic* incompatibilities among independently developed components, which is not solved by any domain independent infrastructure or integration engine. To understand the problems with systems integration it is important to distinguish semantic heterogeneity on the *type level* (in particular manifested in the database schema of a system) from semantic heterogeneity on the *entry level* (corresponding to database contents) (see also [38,39]). Semantic incompatibilities occur at both levels, but the basic problem to be solved in the context of systems integration is the type level heterogeneity. Semantic incompatibilities on the *entry level* are emerging at runtime, when different users enter semantically incompatible data into the system. A typical approach to avoid this kind of heterogeneity is to use a controlled terminology, standard catalogs, or classifications like ICD and ICPM to give the end user a terminological reference. Medical entities dictionaries and terminology servers are to be mentioned in this context (see also [39—41]). Without any terminological control, semantic heterogeneity on the entry level even occurs if there is only a single database, a single application but multiple users with multiple coding conventions. To support data compatibility on the data entry level, ICD and ICPM coding, as well as large catalogs are used in Marburg throughout the university hospital.

Semantic heterogeneity on the *type level* is different, because type level inconsistencies are not made at runtime but at design time. Wherever semantics is ''*hardcoded*'' into an application we have a problem on the type level and a problem with systems integration—this occurs if two different programmers (or vendors) code different and incompatible concepts into their database schemas and application code. Incompatible database schemas will necessarily result if two programmers independently design their database schema without any common ontological basis—none of the two can hopefully guess which concepts the other one will come up with. A posteriori matching of semantically heterogeneous database schemas is a fundamental problem in systems integration which cannot be fully automated [42,43], and in some cases is not even possible without modifying at least one of the database schemas [44]. Semantic heterogeneity on the type level can be avoided or reduced by providing common ontologies as semantic reference for the programmers of disparate systems. Implementing standards like HL7 [45], CCOW [46], or DICOM [47] forces the programmer to adhere to the ontology and business rules that these standards are built upon. So, these healthcare-specific standards did improve the situation, but they do not yet provide commonly accepted comprehensive ontologies and a common application framework which are needed for open component-based systems [48].

Today, integrating heterogeneous and autonomously developed components in healthcare is still a difficult task, and interfaces are still expensive to build and maintain. One of the core problems is still that bridging semantic incompatibilities necessarily has to be done manually which requires a high effort (e.g. see [44,49,50]).

A holistic approach essentially avoids the problem of type level heterogeneity: the programmer creating new applications can refer to the existing concepts (reflected in one common database schema) and does not have to handle semantic heterogeneity stemming from independently designed applications (see also [39]). It could be discussed that controlling entry level heterogeneity by providing catalogs and controlled vocabulary is also somewhat easier to handle in a holistic approach than in a best-of-breed approach. Packaged Enterprise Resource Planning (ERP) systems offer a holistic solution and typically provide a high degree of integration as they use a single central database that follows the old database principle ''one fact in one place'' [51]. Unfortunately, the attempt to map business rules and entities of a complex organization to a single database schema will necessarily result in very large and complex database schemas. As a consequence, large ERP systems have a tendency to evolve slowly and typically cannot live up with all expectations and functional requirements of a large organization. Further disadvantages are:

- Introducing an ERP system is often done in a ''big bang'' approach with large organizational changes. This implies a high project risk and has often led to project failures in the past.
- The evolution of the system depends on the available components of a single vendor, which limits the possibilities of demand-driven system evolution in a specific setting. Software adaptations often lead to long development cycles or high effort custom programming.
- Adaptation to standardized processes coded into an ERP solution bares the risk of losing competitive advantages.
- The commitment to a single vendor also implies an increased risk, because of the dependency of economic development and long term market position of this vendor.

Different vendors increasingly recognize these disadvantages and try to make their systems more flexible by offering standard interfaces for subsystems from other vendors as well as development tools for implementing integrated applications (e.g. see [52]) and incorporating workflow engines into their products (e.g. see [53]).

The layered approach presented in this paper is an attempt to avoid most of the previously mentioned disadvantages of ERP systems by going a step further and adding another layer of abstraction which brings application development closer to the end user and shortens development cycles, which is combined with the attempt to reduce the complexity of developing integrated applications. The approach is used for both custom-made and generic modules that are intended for adaptation and reuse in multiple settings. Thus, the layered approach for continuous system evolution tries to achieve both a high degree of flexibility and of system integrity at the same time. This might contribute to reduce the number of loosely coupled subsystems, as it becomes easier to rapidly implement and adapt integrated components according to end users' needs. Today, a tight coupling of different layers, as proposed in this article, is only possible with proprietary vendor-specific layering, because standard interfaces for layered health information systems are not available yet. The high dependency on one vendor is still a project risk that should not be underestimated. In particular, there is a high risk if framework stability and robustness is insufficient. Therefore, testing is of paramount importance. In our hospital we have developed a catalog of tests which are performed with each new release and application that comes from the vendor—errors detected are removed in close cooperation with the vendor. ERP vendors used to have a tendency to ballast their core systems with a broad spectrum of specific functionality. Hopefully, future systems will put more emphasis on a small but stable and robust core functionality.

To improve the layered approach it is highly desirable to close the gap between process modeling and application implementation. Smith and Fingar have recognized that ''the traditional software development itself creates a discrepancy between business and IT models which are virtually impossible to keep aligned because the relationships between requirements and the associated technical artifacts—components, files, interfaces—are extremely complex and numerous'' [3]. Implementing *paper-like* electronic documents is already close to clinical work practice, but process modeling and implementation still tend to diverge as long as they are separated—the goal should be that the process model *is* the implementation. With the emergence of such tools, application development and process management are about to grow together. The question arises whether *end users* should use such ''process management tools'' to design and improve their own processes. We understand process improvement as the task of conjointly restructuring human work tasks, rearranging organizational routines, redesigning paper forms, and—as one facet — introducing new IT functionalities [1]. The development of these IT functionalities should remain in the hands of IT specialists, even when a generator tool or ''process management tool'' is used. To our experience, physicians want usable systems that seamlessly fit into their work practice, but they do not want to bother configuring and adapting these systems. Application developers, in any case, will still need informatics know how, even if they only deal with process-related artifacts. For example, the application programmer must choose the correct data replication and synchronization strategy according to the semantics of the application: in some cases it is necessary to avoid diverging instances of the same data, in other cases it is necessary to avoid undesired updates on validated documents.

Yet, there is a clear shift of skills within the different roles in the layered approach presented. With a robust underlying framework, the application developer can concentrate on business process alignment instead of coding and debugging. Thus, the approach presented, fosters the domain specialization of application developers, and it is a step towards a responsive IT infrastructure alleviating a demand-driven introduction of IT functionality in the context of cooperative process improvement.

References

[1] M. Berg, P. Toussaint, The mantra of modeling and the forgotten powers of paper: a sociotechnical view on the development of process-oriented ICT in health care, Int. J. Med. Inform. 69 (2/3) (2003) 223—234.
[2] K.A. Kuhn, R. Lenz, T. Elstner, H. Siegele, R. Moll, Experiences with a generator tool for building clinical application modules, Methods Inform. Med. 42 (1) (2003) 37—44.
[3] H. Smith, P. Fingar, Business Process Management: The Third Wave, 1st ed., Meghan-Kiffer Press, London, 2002.
[4] M. Al-Mashari, M. Zairi, BPR implementation process: an analysis of key success and failure factors, Business Process Manage. J. 5 (1) (1999) 87—112.
[5] C. Sauer, Deciding the future for IS failures: not the choice you might think, in: W. Curie, R. Galliers (Eds.), Rethinking Management Information Systems, Oxford University Press, Oxford, 1999, pp. 279—309.
[6] G. Versteegen, Projektmanagement mit dem Rational Unified Process, Springer-Verlag, Berlin, 2000.
[7] P. Kruchten, The Rational Unified Process: An Introduction, Addison-Wesley, Canada, 2000.
[8] K. Beck, Extreme Programming, Addison-Wesley, München, 2000.
[9] M. Hammer, J. Champy, Reengineering the Corporation a Manifesto for Business Revolution, 1st ed., HarperBusiness, New York, 1993.
[10] T.H. Davenport, Process Innovation Reengineering Work through Information Technology, Harvard Business School Press, Boston, MA, 1993.
[11] J. Caron, S. Jarvenpaa, D. Stoddard, Business reengineering at CIGNA corporation: experiences and lessons learned from the first five years, MIS Quart. 18 (3) (1994) 233—250.
[12] E. Murphy, Cultural values, workplace democracy and organisation: emerging issues in European businesses, in: C. Coulson-Thomas (Ed.), Business Process Re-engineering: Myth & Reality, Kogan Page, London, 1994, pp. 201—210.
[13] C.J. Atkinson, V.J. Peel, Transforming a hospital through growing, not building, an electronic patient record system, Methods Inform. Med. 37 (3) (1998) 285—293.
[14] M. Fayad, D.C. Schmidt, R. Johnson, Implementing Application Frameworks Object-Oriented Frameworks at Work, Wiley, New York, 1999.
[15] M. Fayad, R. Johnson, Domain-Specific Application Frameworks Experienced by Industry, Wiley, New York, 2000.
[16] G. Coleman, R. Verbruggen, A quality software process for rapid application development, Software Qual. J. 1998 (7) (1998) 107—122.
[17] J. Brender, Methodology for constructive assessment of IT-based systems in an organisational context, Int. J. Med. Inform. 56 (1—3) (1999) 67—86.
[18] C. Sjoberg, T. Timpka, Participatory design of information systems in health care, J. Am. Med. Inform. Assoc. 5 (2) (1998) 177—183.
[19] R.D. Ellis, T.B. Jankowski, J.E. Jasper, Participatory design of an Internet-based information system for aging services professionals, Gerontologist 38 (6) (1998) 743—748.
[20] G. Booch, J. Rumbaugh, I. Jacobson, The Unified Modeling Language User Guide, Addison-Wesley, Reading, MA, 1999.
[21] A.S. Tanenbaum, Modern Operating Systems, Prentice-Hall, Englewood Cliffs, NJ, 1992.
[22] T. Härder, Realisierung von operationalen Schnittstellen, in: P.C. Lockemann, J.W. Schmidt (Eds.), Datenbank-Handbuch, Springer-Verlag, Berlin, 1987.
[23] A.S. Tanenbaum, Computer Networks, 2nd ed., Prentice-Hall, Englewood Cliffs, NJ, 1988.
[24] R. Lenz, T. Elstner, H. Siegele, K.A. Kuhn, A practical approach to process support in health information systems, J. Am. Med. Inform. Assoc. 9 (6) (2002) 571—585.

[25] M. Reichert, P. Dadam, Geschäftsprozessmodellierung und workflow-management—konzepte, systeme und deren anwendung, Ind. Manage. 2000 (3) (2000) 23—27.
[26] M. Beyer, W. Hesse, Einsatz von UML zur software-prozeßmodellierung, GI-Softwaretechnik-Trends 22 (1) (2002) 4—11.
[27] J.S. Ash, P.Z. Stavri, R. Dykstra, L. Fournier, Implementing computerized physician order entry: the importance of special people, Int. J. Med. Inform. 69 (2/3) (2003) 235—250.
[28] S. Groothuis, G.G. van Merode, Discrete event simulation in the health policy and management program, Methods Inform. Med. 39 (4/5) (2000) 339—342.
[29] S. Groothuis, G.G. van Merode, A. Hasman, Simulation as decision tool for capacity planning, Comput. Methods Programs Biomed. 66 (2/3) (2001) 139—151.
[30] G. Willis, Failure modes and effects analysis in clinical engineering, J. Clin. Eng. 17 (1) (1992) 59—63.
[31] J. Burgmeier, Failure mode and effect analysis: an application in reducing risk in blood transfusion, Jt. Comm. J. Qual. Improv. 28 (6) (2002) 331—339.
[32] R. Lenz, T. Elstner, K. Kuhn, Experiences with a holistic information system, Proc. AMIA Symp. (2001) 952.
[33] P.D. Clayton, S.P. Narus, S.M. Huff, T.A. Pryor, P.J. Haug, T. Larkin, et al., Building a comprehensive clinical information system from components. The approach at Intermountain Health Care, Methods Inform. Med. 42 (1) (2003) 1—7.
[34] R. van de Velde, Framework for a clinical information system, Int. J. Med. Inform. 57 (1) (2000) 57—72.
[35] P. Degoulet, L. Marin, M. Lavril, C. Le Bozec, E. Delbecke, J.J. Meaux, et al., The HEGP component-based clinical information system, Int. J. Med. Inform. 69 (2/3) (2003) 115—126.
[36] V. Gruhn, A. Thiel, Komponentenmodelle—DCOM, JavaBeans, Enterprise JavaBeans, CORBA, 1st ed., Addison-Wesley, München, 2000.
[37] C. Szyperski, Component Software—Beyond Object-Oriented Programming, 1st ed., Addison-Wesley, Harlow, UK, 1999.
[38] M.A. Musen, Domain ontologies in software engineering: use of Protege with the EON architecture, Methods Inform. Med. 37 (4/5) (1998) 540—550.
[39] K.A. Kuhn, D.A. Giuse, From hospital information systems to health information systems. Problems, challenges, perspectives, Methods Inform. Med. 40 (4) (2001) 275—287.
[40] J.J. Cimino, From data to knowledge through concept-oriented terminologies: experience with the Medical Entities Dictionary, J. Am. Med. Inform. Assoc. 7 (3) (2000) 288—297.
[41] W.A. Nowlan, A.L. Rector, T.W. Rush, W.D. Solomon, From terminology to terminology services, Proc. Annu. Symp. Comput. Appl. Med. Care 150 (4) (1994) 150—154.
[42] A. Elmagarmid, M. Rusinkiewicz, A. Sheth, Management of Heterogeneous and Autonomous Database Systems, Morgan Kaufmann Publishers, San Francisco, CA, 1999.
[43] E. Rahm, P.A. Bernstein, A survey of approaches to automatic schema matching, VLDB J. 2001 (10) (2001) 334—350.
[44] R.M. Colomb, Impact of semantic heterogeneity on federating databases, Comput. J. 40 (5) (1997) 235—244.
[45] G.W. Beeler, HL7 version 3—an object-oriented methodology for collaborative standards development, Int. J. Med. Inform. 48 (1—3) (1998) 151—161.
[46] Seliger R., Overview of HL7's CCOW Standard, Health Level Seven, Inc. (2001): http://www.hl7.org/library/committees/sigvi/ccow_overview_2001.doc.
[47] W.D. Bidgood, S.C. Horii, F.W. Prior, D.E. Van Syckle, Understanding and using DICOM, the data interchange standard for biomedical imaging, J. Am. Med. Inform. Assoc. 4 (3) (1997) 199—212.
[48] R. Lenz, S. Huff, A. Geissbühler, Report of conference track 2: pathways to open architectures, Int. J. Med. Inform. 69 (2/3) (2003) 297—299.
[49] D. McGoveran, Business semantics, EAI J. (October) (2000) 10.
[50] M. Stonebraker, Integrating islands of information, EAI J. (September/October) (1999) 1—5.
[51] C.J. Date, An Introduction to Database Systems, 6th ed., Addison-Wesley, Reading, MA, 1995.
[52] G. Gell, P. Schmucker, M. Pedevilla, H. Leitner, J. Naumann, H. Fuchs, et al., SAP and partners: IS-H and IS-H* MED, Methods Inform. Med. 42 (1) (2003) 16—24.
[53] R. Haux, C. Seggewies, W. Baldauf-Sobez, P. Kullmann, H. Reichert, L. Luedecke, et al., Soarian—workflow management applied for health care, Methods Inform. Med. 42 (1) (2003) 25—36.